Machine Learning Course - CS-433

Generative Models

Dec 6, 2022

Martin Jaggi
Last updated on: December 6, 2022

credits to Tatjana Chavdarova and Lara Orlandic

Motivation

Generative models model a probability distribution over a set of random variables either *explicitly* or *implicitly*. In the latter case, we do not have direct access to the underlying probability distribution, but we can sample according to it.

Generative Adversarial Networks (GANs, Goodfellow et al. [2014]) are a family of implicit generative algorithms that are fast to sample from. Contrary to single-objective minimization $f: \mathcal{X} \to \mathbb{R}$, The optimization of a GAN is formulated as a differentiable two-player game where the generator G with parameters $\boldsymbol{\theta}$, and the discriminator D with parameters $\boldsymbol{\varphi}$, aim at minimizing their own cost function $\mathcal{L}^{\boldsymbol{\theta}}$ and $\mathcal{L}^{\boldsymbol{\varphi}}$, respectively, as follows:

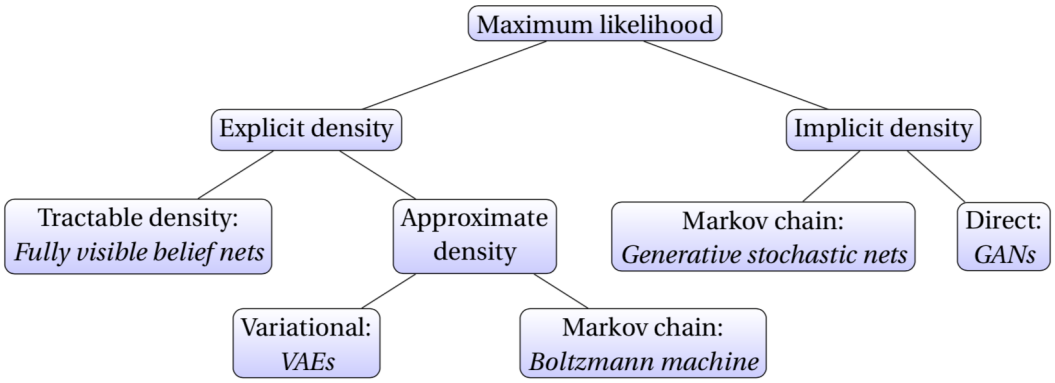
$$\begin{aligned} \boldsymbol{\theta}^\star &\in \arg\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}^\star) \\ \boldsymbol{\varphi}^\star &\in \arg\min_{\boldsymbol{\varphi}\in\Phi} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}) \,. \end{aligned} \tag{2P-G}$$

When $\mathcal{L}^{\boldsymbol{\theta}} = -\mathcal{L}^{\boldsymbol{\varphi}}$ the game is called a zero-sum game and (2P-G) is a minmax problem.

Generative Models

Given a data sample x, a *discriminative* model aims at predicting its label y, hence it models the posterior distribution $p(y|x)$. Generative models instead model the distribution $p(x)$ defined over the datapoints x. Depending on the type of the generative model we can either evaluate the probability assigned to each datapoint, or sample according to it.

Taxonomy. The following figure depicts the taxonomy of the existing generative methods that are based on maximum likelihood:



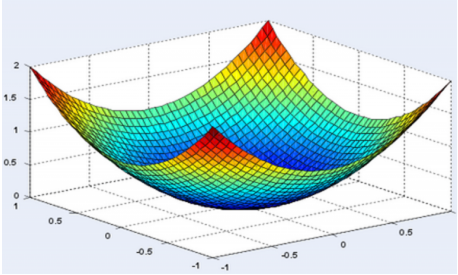
(the above figure is generated according to Goodfellow [2016])

The *explicit density* generative models explicitly model the distribution that describes the probability that the model assigns to each datapoint, controlled by parameters θ. These models can be further categorized as: (i) *exact* or *tractable*, and (ii) *approximate* models. Rather than calculating the likelihood, *implicit density models* instead provide a way to draw samples.

Applications. Due to the high dimensionality of the raw data of real-world tasks, a critical step for efficiency (memory & computation) is learning a semantically meaningful subspace. Generative models share the same underlying goal of learning representations. Other applications include: (i) *planning* in Reinforcement learning [Sutton and Barto, 2018]–where agents simulate sequences of outcomes [*e.g.* Kurutach et al., 2018], (ii) physics–*e.g.* generating plausible trajectory of a Hamiltonian particle [Greydanus et al., 2019], among many others.

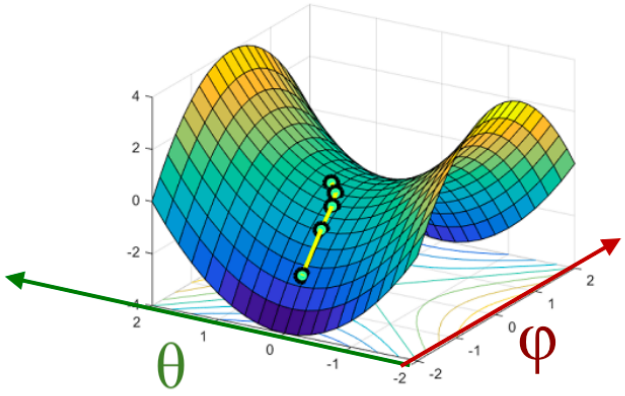
2-Player vs. Single-objective minimization

1. Standard supervised learning: convex objective



2. Minmax: convex–concave objective

$$\min_{\theta} \max_{\varphi} L(\theta, \varphi)$$



(The above figure is adjusted from Vaishnavh Nagarajan)

We would like to converge to a point called Nash equilibrium (NE). In the context of game theory, NE is a combination of strategies from which, no player has an incentive to deviate unilaterally.

Differential Nash Equilibrium

More formally, a Nash equilibrium for a continuous game is defined as a point $(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)$ where:

$$\mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}) \leq \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star) \leq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}^\star) \; \forall \boldsymbol{\theta}, \boldsymbol{\varphi} \,. \tag{NE}$$

Such points are (locally) optimal for both players with respect to their own decision variable, *i.e.* no player has the incentive to unilaterally deviate from it.

In machine learning we are interested in differential games where $\mathcal{L}$ is twice differentiable, in which case such NE needs to satisfy slightly stronger conditions. A point $(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)$ is a Differential Nash Equilibrium (DNE) of a zero-sum game iff:

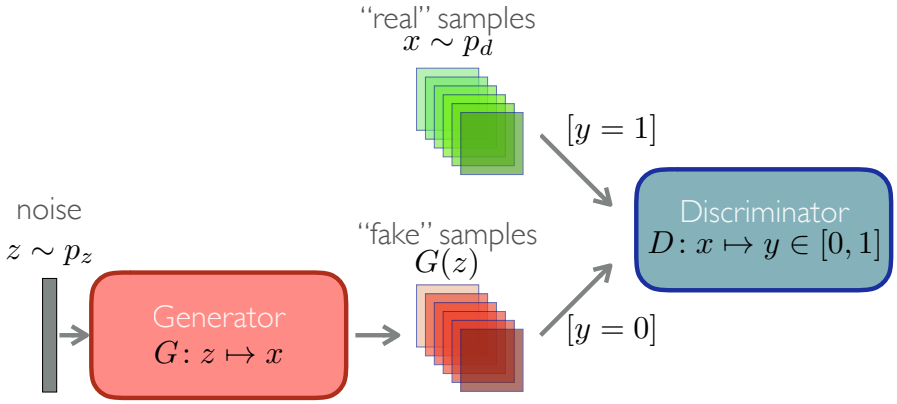
$$\begin{aligned} \|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)\| = \|\nabla_{\boldsymbol{\varphi}} \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)\| &= 0, \\ \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star) &\succ 0, \text{ and} \\ \nabla^2_{\boldsymbol{\varphi}} \mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star) &\prec 0 \,, \end{aligned} \tag{DNE}$$

where $A \succ 0$ and $A \prec 0$ iff A is positive definite and negative definite, respectively[a].

[a] Note that the key difference between DNE and NE is that $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\cdot)$ and $\nabla^2_{\boldsymbol{\varphi}} \mathcal{L}(\cdot)$ for DNE are required to be definite (instead of semidefinite).

Generative Adversarial Networks

1. The discriminator "distinguishes"
   real vs. *fake* samples:

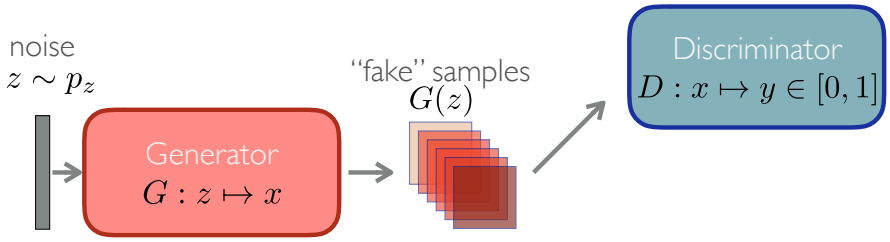


   p_z known "noise" distribution, *e.g.* $\mathcal{N}(0, 1)$

   p_d the real data distribution

   D mapping $D: x \mapsto y \in [0, 1]$,
   where y is an estimated probability that $x \sim p_d$

2. The generator aims at fooling
   the discriminator that its samples are real:



   G mapping $G: z \mapsto x$, such that if $z \sim p_z$,
   then hopefully $x \sim p_d$

   p_g the "fake" data distribution

3. Objective

$$\min_G \max_D \mathop{\mathbb{E}}_{x\sim p_d}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z}[\log(1-D(G(z)))]$$

- Loss for D: distinguish between $x \sim p_g$ and $x \sim p_d$ (binary classification):

$$\mathscr{L}_D(G, D) = \max_D \mathop{\mathbb{E}}_{x\sim p_d}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z}[\log(1-D(G(z)))]$$

- Loss for G: fool D that $G(z) \sim p_d$:

$$\begin{aligned}\mathscr{L}_G(G, D) &= \min_G \mathop{\mathbb{E}}_{z\sim p_z}[\log(1 - D(G(z)))] \\ &:= \text{(in practice)} \max_G \mathop{\mathbb{E}}_{z\sim p_z}[\log(D(G(z)))]\end{aligned}$$

4. **Theoretical Solution:** The optimum is reached when $p_g = p_d$ and the optimal value is $-\log 4$ (proof in function space, see next slides).

KL and JS divergences

Before proving that at the equilibrium of the above GAN framework $p_g = p_d$ we need to define some measures of similarity between two probability distributions: The KL and JS divergences.

The Kullback–Leibler (KL) divergence is defined as:

$$\mathbb{D}_{KL}(p_d||p_g) := \int_x \log\left(\frac{p_d(x)}{p_g(x)}\right) p_d(x)\,\mathrm{d}x\,.$$

KL is also called *relative entropy*, as it measures how one probability distribution is different from a "reference" probability distribution, and it is asymmetric.

The Jenson-Shannon (JS) divergence is defined as:

$$\mathbb{D}_{JS}(p\|q) := \frac{1}{2}\mathbb{D}_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2}\mathbb{D}_{KL}(q\|\frac{p+q}{2})$$

Note that contrary to the KL divergence defined above, the JS divergence is symmetric.

The GAN framework: Equilibrium at $p_g = p_d$

In the following, we'll assume the neural network models G and D have infinite capacity, so can represent any probability distribution. We will study the convergence of the loss function in the space of probability density functions.

The discriminator maximizes:

$$\begin{aligned}\mathcal{L}(G, D) &= \int_x p_d(x) \log(D(x))\, \mathrm{d}x \\ &\quad + \int_z p_z(z) \log(1 - D(G(z)))\, \mathrm{d}z \\ &= \int_x p_d(x) \log(D(x)) + p_g(x) \log(1 - D(x))\, \mathrm{d}x\end{aligned}$$

Where we used $x = G(z)$, and p_g is the distribution of x.
The above integrand can be written as: $f(y) = a \log y + b \log(1 - y)$. To solve for its critical points: $f'(y) = 0 \Rightarrow \frac{a}{y} - \frac{b}{1-y} = 0 \Rightarrow y = \frac{a}{a+b}$. Moreover, if $a + b \neq 0$ we obtain that $\frac{a}{a+b}$ is a maximum as $f''(\frac{a}{a+b}) < 0$.
Hence, optimal discriminator $D^\star$ is:

$$D^\star(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

By replacing the optimal discriminator in the above objective, we obtain that the generator minimizes:

$$\begin{aligned}\mathcal{L}(G, D^\star) &= \mathop{\mathbb{E}}_{x\sim p_d}[\log D^\star(x)] + \mathop{\mathbb{E}}_{x\sim p_g}[\log(1 - D^\star(x))] \\ &= \mathop{\mathbb{E}}_{x\sim p_d}[\log \frac{p_d(x)}{p_d(x) + p_g(x)}] + \mathop{\mathbb{E}}_{x\sim p_g}[\log \frac{p_g(x)}{p_d(x) + p_g(x)}] \\ &= -\log 4 + \mathbb{D}_{KL}\Big(p_d \Big\| \frac{p_d + p_g}{2}\Big) + \mathbb{D}_{KL}\Big(p_g \Big\| \frac{p_d + p_g}{2}\Big) \\ &= -\log 4 + 2 \cdot \mathbb{D}_{JS}(p_d \| p_g)\end{aligned}$$

where to obtain the third expression we used the definition of logarithm:

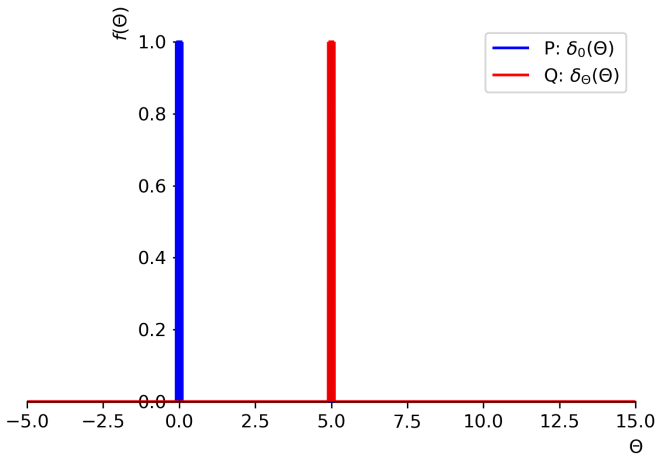
$$\begin{aligned}&\log 2 + \log\left(\frac{p_d(x)}{p_g(x) + p_d(x)}\right) \\ &= \log\left(2\frac{p_d(x)}{p_g(x) + p_d(x)}\right) \\ &= \log\left(\frac{p_d(x)}{\frac{p_g(x)+p_d(x)}{2}}\right).\end{aligned}$$

Above, $\mathbb{D}_{KL}$ and $\mathbb{D}_{JS}$ again denote the Kullback–Leibler and the Jenson-Shannon divergences (see previous slides).

The optimum is reached when $p_g = p_d$ (note $D^\star = \frac{1}{2}$), and the optimal value is $-\log 4$.

Drawback of using JS divergence for GANs

Example: Let us consider two probability distributions defined on $\mathbb{R}$: P: $\delta_0(x)$ and Q: $\delta_\theta(x)$.



Note that when the supports of the two distributions are disjoint ($\theta \neq 0$), we obtain $\mathbb{D}_{KL}(P||Q) = +\infty$, and $\mathbb{D}_{JS}(P||Q) = \log 2$, both yielding non-smooth gradient, making gradient-based methods hard to use [Arjovsky et al., 2017].

Wasserstein Distance

The previous example motivates the use of the Wasserstein distance (*aka Earth mover's* distance) in the context of GANs, described next.

Wasserstein-1 distance is defined as:

$$\mathbb{D}_W(p_d, p_g) = \inf_{\gamma \sim \Pi(p_d, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|],$$

where $\Pi(p_d, p_g)$ is the set of all possible joint probability distributions between p_d and p_g, whose marginals are p_d, p_g, resp.

Intuitively, the two distributions can be viewed as a mass on each point. The goal is to move these masses so that one distribution can be transformed into the other. As there are infinitely many ways of doing so, $\mathbb{D}_W(p_d, p_g)$ is the *minimum cost* we need to spend. The cost is the amount of mass that has to be transported times the distance it has to be moved.

Note that in the above example $\mathbb{D}_W(\delta_0(x), \delta_\theta(x))=|\theta|$, and it provides "usable" gradient. However, $\mathbb{D}_W(\cdot)$ does not scale with the dimensionality of the input random variables, as the number of states of the joint probability distribution grows exponentially. Fortunately, it can be alternatively formalized (see next slide).

Wasserstein GAN

(optional material)

Def. $f : \mathbb{R} \to \mathbb{R}$ is called k-Lipschitz continuous if $\exists k \in \mathbb{R}$ s.t.

$$|f(x_1) - f(x_2)| \leq k|x_1 - x_2|, \quad \forall x_1, x_2.$$

The so called Wasserstein GAN [WGAN, Arjovsky et al., 2017, Gulrajani et al., 2017] replaces the JS divergence with the Wasserstein distance. As the Wasserstein distance is intractable for Deep Neural Nets, WGANs make use of the so called Kantorovich-Rubinstein duality principle, which tells us that:

$$\mathbb{D}_W(p_d, p_g) = \sup_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{x \sim p_d} [f(x)] - \mathop{\mathbb{E}}_{x \sim p_g} [f(x)],$$

where the supremum is over 1-Lipschitz functions $f : \mathcal{X} \mapsto \mathbb{R}$. Its derivation is out of the scope of this course (if interested see proof sketch in the Appendix).
In the context of GANs, f is the function represented by the discriminator (called *critic* in WGAN), yielding:

$$\min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{x \sim p_d} [D(x)] - \mathop{\mathbb{E}}_{x \sim p_g} [D(x)],$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions (see next slide).

Some GAN variants with Lipschitz Discriminator

(optional material)

The constraint that the discriminator should be 1-Lipschitz can be enforced in several ways. The table below summarizes some of the GAN variants which enforce such constraint.

WGAN [Arjovsky et al., 2017] uses straightforward weight clipping. [Gulrajani et al., 2017] point out that this may lead to optimization difficulties, and proposed adding an extra penalty term to the training loss of the Discriminator, which penalizes gradients whose norm is higher than 1. As enforcing this for any input is intractable, this is done by considering a line between fake and real samples, obtained by interpolating between these. [Kodali et al., 2017] penalize gradients whose norm is higher than 1 while considering samples in a region around real data points. Note that *DRAGAN* & *WGAN−GP* are computationally more expensive as each parameter update requires computation of a second-order derivative (as the loss includes gradient penalty). [Miyato et al., 2018] make use of the power iteration method to estimate the largest singular value per layer, so as to divide the parameters of that layer with it. In the context of training GANs, although the power iteration method requires multiple iterations to compute the largest singular value, it cam be implemented efficiently due to the fact that it is combined with Stochastic gradient based method. In other words, it was shown in practice that one update of it per stochastic parameter update is sufficient to obtain good estimates of the largest singular value. It was empirically shown that enforcing 1-Lipschitz Discriminator helps the convergence of GAN trained with JS-based losses, and JS-based GANs remained widely used in practice. Moreover, besides lacking theoretical backup, one of the well performing large-scale GAN variants further enforces that both the Generator and the discriminator are 1-Lipschitz functions.

	$\mathbb{D}\_(p_d\|\|p_g)$	G/D	Mean of enforcing Lipschitz continuity
WGAN [Arjovsky et al., 2017]	$\mathbb{D}_W$	D	Weight clipping: $w = clip(w, -c, c)$
WGAN−GP [Gulrajani et al., 2017]	$\mathbb{D}_W$	D	Gradient penalty: line between real and fake points
DRAGAN [Kodali et al., 2017]	$\sim \mathbb{D}_{JSD}$	D	Gradient penalty: around real data points
SNGAN [Miyato et al., 2018]	$\sim \mathbb{D}_{JSD}$	D	Spectral norm using the power iteration method
bigGAN [Brock et al., 2019]	$\sim \mathbb{D}_{JSD}$	$G\&D$	Spectral norm using the power iteration method

Alternating–GAN algorithm

In practice, G and D are parametrized models (typically neural networks), and are optimized with gradient based methods.

- G: deep neural network $G(\mathbf{z}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$
- D: deep neural network $D(\mathbf{x}; \boldsymbol{\varphi})$ with parameters $\boldsymbol{\varphi}$

In most GAN implementations G and D have different losses $\mathcal{L}^{\boldsymbol{\theta}}$ and $\mathcal{L}^{\boldsymbol{\varphi}}$[a], resp. In the following, we present the most commonly used algorithm for training GANs.

Algorithm 1 *alternating*–GAN

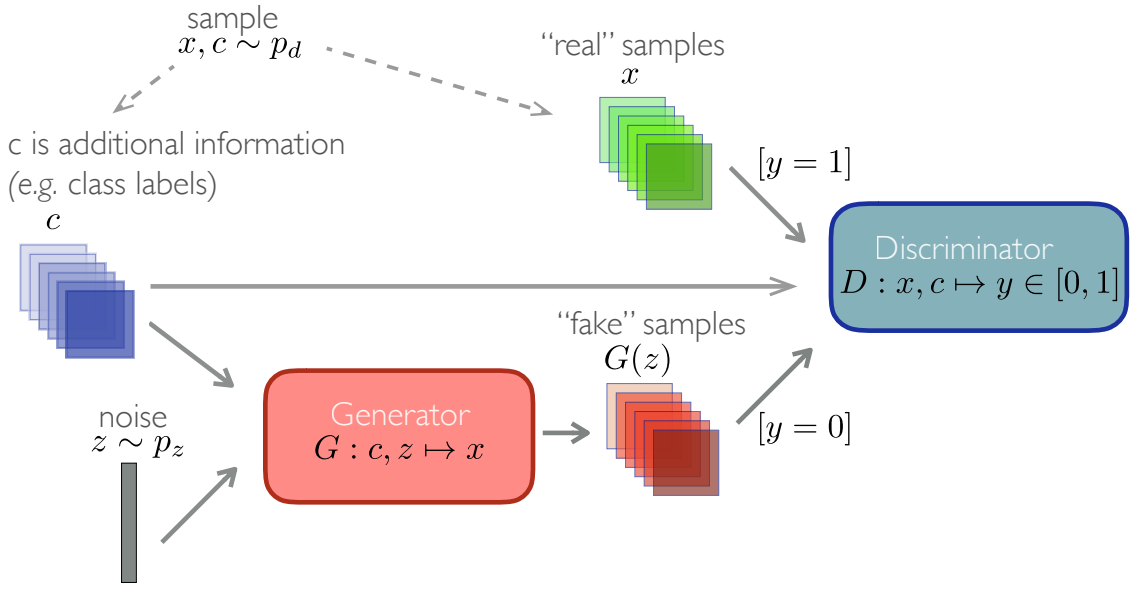
Input: dataset $\mathcal{D}$, known distribution p_z, learning rate η, generator loss $\mathcal{L}^{\boldsymbol{\theta}}$, discriminator loss $\mathcal{L}^{\boldsymbol{\varphi}}$
Initialize: $D(\cdot; \boldsymbol{\varphi})$, $G(\cdot; \boldsymbol{\theta})$
for $t = 1$ **to** T **do**
  $\mathbf{x} \sim p_x$ {sample real data}
  $\mathbf{z} \sim p_z$ {sample noise from p_z}
  $\boldsymbol{\varphi} = \boldsymbol{\varphi} - \eta \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathbf{x}, \mathbf{z})$ {update D}
  $\mathbf{z} \sim p_z$ {sample noise from p_z}
  $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathbf{z})$ {update G}
end for
Output: $\boldsymbol{\theta}, \boldsymbol{\varphi}$

For simplicity, in Alg.1 we used gradient descent, however in practice GANs are often optimized using Adam [Kingma and Ba, 2015], and developing well performing minimax optimization methods is an active research area.

[a] Although we *minimize* both the loses, this notation generalizes the zero-sum game, as the latter holds when $\mathcal{L}^{\boldsymbol{\theta}} = -\mathcal{L}^{\boldsymbol{\varphi}} := \mathcal{L}$.

Conditional GAN – (CGAN)

Many applications require generative model of a conditional probability distribution (*e.g.* "in-painting", segmentation, predicting the next frame of a video *etc.*).
GANs can also generate samples of conditional distribution, called Conditional GAN (CGAN) [Mirza and Osindero, 2014]. In CGANs both the Generator and the Discriminator are conditioned during training by some additional information, typically the class labels (but could also be images *e.g.* auto-generated edges of an image, conditioning on non-occluded portion–so as to generate the occluded part of the image *etc*).



When conditioning on the class labels, typically one-hot vector representation of the class labels is used (empirically shown to perform better).

GAN architectures for images

In the context of images synthesis, [Radford et al., 2015] propose specific architectures of the two models, named Deep Convolutional GANs – *DCGAN*. Notably, the generator uses *transposed convolutional layer* (*a.k.a.* fractionally strided convolutions), also informally called "Deconvolution layers" (wrongfully). Simlest way to explain these is that they "swap" the forward and the backward passes of a convolution layer: the forward transposed convolution operation can be thought of as the gradient of some convolution with respect to its input, which is usually how transposed convolutions are also implemented in practice.

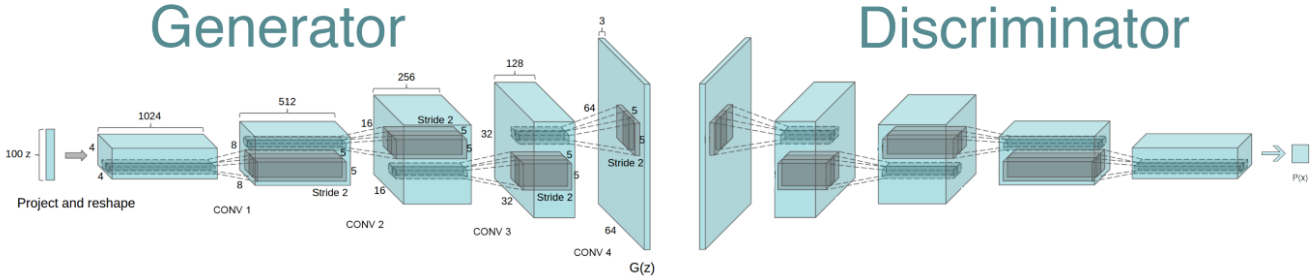
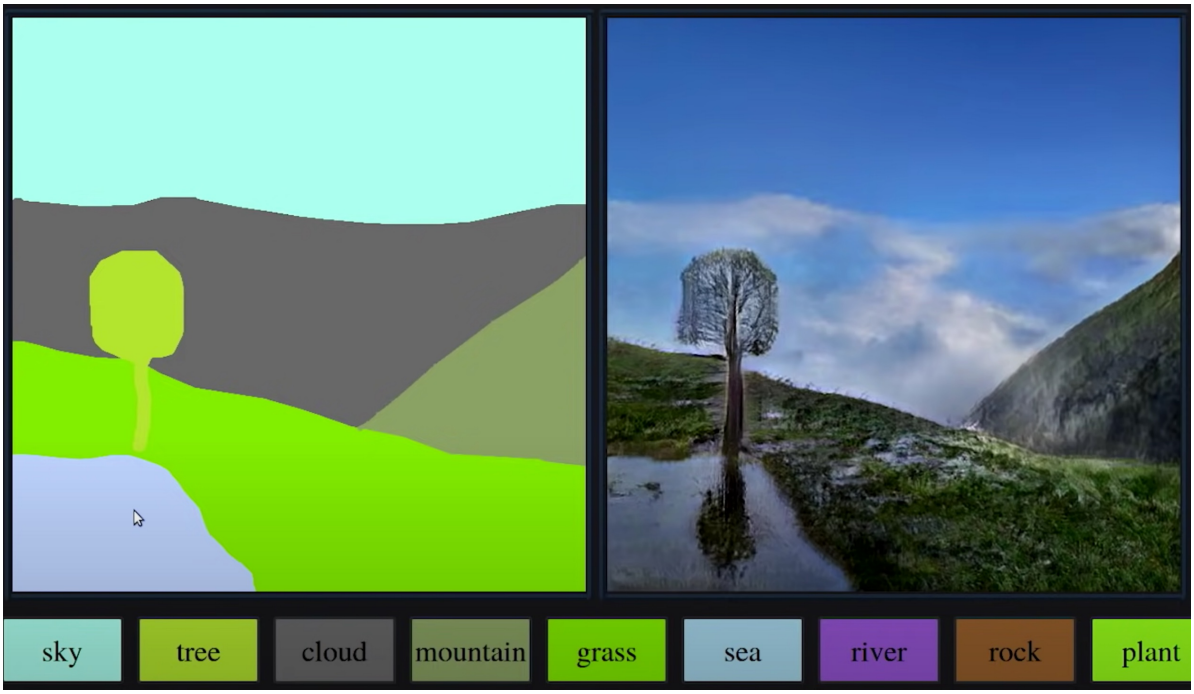


Image to image translation

Isola et al. (2016), (CGAN): automatically detected edges ↦ handbags

Isola et al. (2016), generalization of the CGAN model trained on edges ↦ photo (see previous figure) to human-drawn sketches

Conditional Generation of Images



(Picture from blogs.nvidia.com/blog/2021/11/22/gaugan2-ai-art-demo/.)

"A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019, https://arxiv.org/abs/1812.04948

CycleGAN

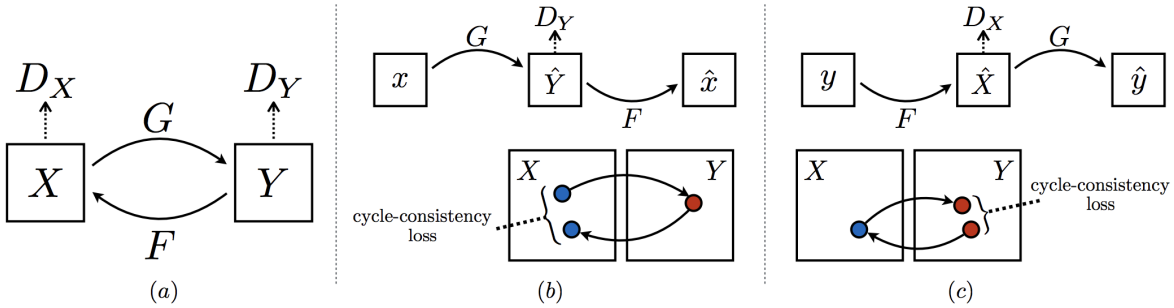


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

(The above figure is taken from Zhu et al. [2017].)

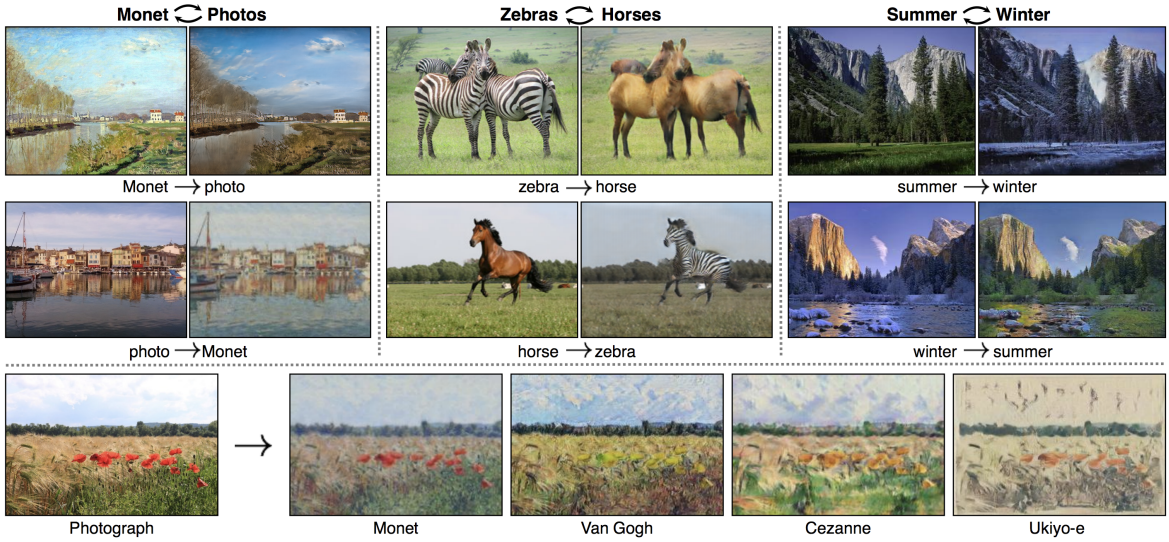
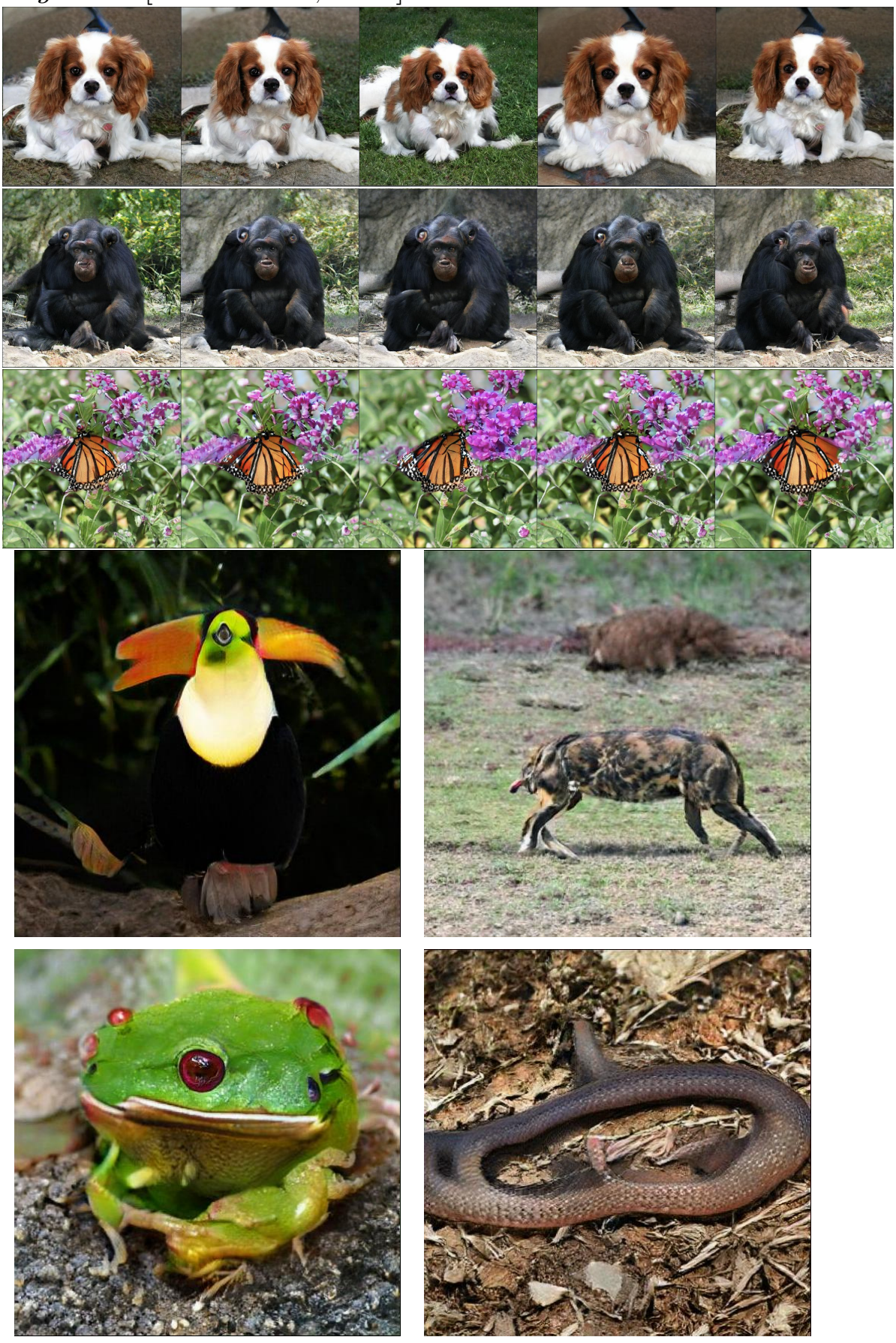


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

(The above figure is taken from Zhu et al. [2017].)

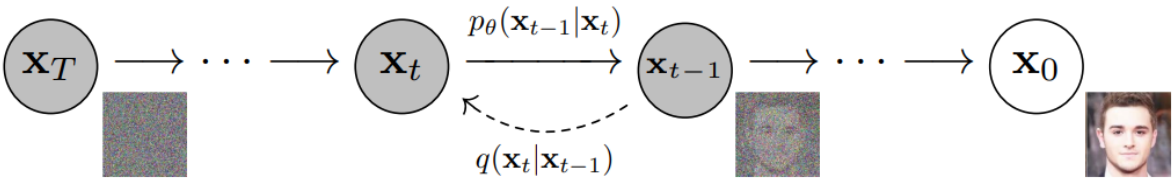
GAN generated images

512×512 samples from the class-conditional *BigGAN* [Brock et al., 2019].



Diffusion models

Diffusion models are another class of generative models that have gained popularity in recent years, both for their enhanced performance and implementation in AI image generators like DALL-E 2, Stable Diffusion, and Midjourney. Unlike GANs, which train two separate models for data generation and discrimination, Diffusion Models work by progressively adding noise to input data and training one single model to estimate the added noise and recover the data.



The figure above from Ho et al. [2020] describes the steps of the diffusion process, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are latent representations of the input data $\mathbf{x}_0$ that share its dimensions. The forward process (right to left in the image above) consists of a Markov chain that adds Gaussian noise $\mathbf{q}$ with a variance β_t to each latent $\mathbf{x}_{t-1}$:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \boldsymbol{I}).$$

The reverse process (left to right in the image above) learns the transitions of the Markov chain through the noise distribution p, where:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)).$$

A model is trained to estimate the added noise at each step of the process, and a simple L2 loss function compares the predicted noise to the actual added noise. Then, data can be generated starting from random inputs by modeling the noise at each step and subtracting it from the image.

Instead of starting from a purely random image, DALL-E 2 begins by embedding the inputted text and transforming it into an image, which is then decoded using a Diffusion model. The text embedding simply provides additional information to condition the input of the Diffusion model. Here are some examples of transformed text provided by Ramesh et al. [2022]:

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a dolphin in an astronaut suit on saturn, artstation

a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese

Applications of GANs and Diffusion Models

- Generating images;
- edges to realistic photos [Isola et al., 2017];
- old gray-scale images to RGB
- Semi supervised learning;
- Text–to–image generation;
- Super resolution;
- Image editing;
- Image Inpainting (filling gaps);
- Adversarial examples (Defense Vs. Attack of classifiers);
- Videos (generation/prediction);
- Domain-transfer;
- Audio;
- Tabular data;
- Also: physics, games...

GANs and diffusion models have also been used for other data modalities: For raw-waveform audio synthesis, examples include WaveGAN [Donahue et al., 2019] and MelGAN [Kumar et al., 2019], among others. For generating realistic tabular data, see e.g. [Kotelnikov et al., 2022].

Summary

We have studied:

1. Zero-sum games & its solution
2. Generative Adversarial Networks
   Players (generator & discriminator), Objectives, Solution & Algorithm
3. KL and JS divergences and Wasserstein distance
   GANs, WGANs
4. Some GAN variants (CGAN & CycleGAN) & applications of (C)GANs
5. Diffusion models & applications

Additional Notes

(optional material)

Appendix A: Wasserstein Distance

Continious probability distributions

The Wasserstein distance between two probability distributions μ and ν is defined as:

$$W(\mu,\nu) = \inf_{\pi(\mu,\nu)} \iint c(x,y)\pi(dx,dy), \tag{1}$$

Equation (1) assumes continuous distributions μ and ν. If we use the Euclidean distance we have:

$$\begin{aligned} W(\mu,\nu) &= \inf_{\pi} \iint \|x-y\|\pi(x,y)\,\mathrm{d}x\,\mathrm{d}y \\ &= \inf_{\pi} \mathbb{E}_{x,y\sim\pi}[\|x-y\|]. \end{aligned} \tag{2}$$

Discrete probability distributions

Lets consider two discrete distributions $\mathbb{P}_x, \mathbb{P}_y$, with s states each: x_i and y_i, $i = 1 \dots s$. Thus,

$$\begin{aligned} W(\mathbb{P}_x,\mathbb{P}_y) &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \sum_{x,y} \|x-y\|\Pi(x,y) \\ &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \mathbb{E}_{(x,y)\sim\Pi}[\|x-y\|] \\ &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \langle \Pi, D\rangle, \end{aligned} \tag{3}$$

where with $\langle\cdot,\cdot\rangle$ we denote sum of element-wise multiplication, and $\Pi \in \mathbb{R}^{s\times s}$ is the joint probability and $D \in \mathbb{R}^{s\times s}$ is the Euclidean distance between each $x_i, y_j, i = 1 \dots s, j = 1 \dots s$.

Kantorovich-Rubinstein dualty principle

From Kantorovich-Rubinstein dualty [Villani, 2008]:

$$\mathbb{D}_W(p_d,p_g) = \frac{1}{K} \sup_{\|f\|_L\le K} \mathop{\mathbb{E}}_{x\sim p_d}[f(x)] - \mathop{\mathbb{E}}_{x\sim p_g}[f(x)]$$

Villani [2008] gives the following intuitive interpretation of the above Kantorovich duality principle. Namely, if our goal is to transfer a huge amount of mass distributed over certain area, to a different distinct area, we would like to minimize the cost for transport, thus we have an inf over the implied cost. Suppose we have a middle-man who offers to handle the problem for us, by claiming that he will not charge us more than the actual transport cost (thus $\varphi(y) - \psi(x) \leq c(x, y)$). Then, our initial problem is in fact equal to the one of the middle-man trying to maximize his profit. His profit on the other hand is defined as a price for loading goods: $\varphi(x)$ and a price for unloading them at destination y: $\psi(y)$. Naturally, he aims at maximizing his profit (thus the sup). Note however that the middle man is ready to give financial compensations for some places, in the form of negative prices.

Proof sketch. See [Villani, 2008] for full formal proof.

x	cont. r.v.
μ	distribution of x
ν	target distribution
$c(\cdot)$	cost, *e.g.* ℓ_p norm

$$\begin{aligned}\mathbb{D}_W(\mu, \nu) &= \inf_{\gamma \in \pi} \iint c(x, y)\, \mathrm{d}\gamma(x, y) \\ &= \sup_{\varphi, \psi \to \mathbb{R}} \int \varphi(y)\, \mathrm{d}\nu - \psi(x)\, \mathrm{d}\mu\end{aligned}$$

where $\varphi(y) - \psi(x) \leq c(x, y)$, and π is set of all non-negative Borel measures, whose marginals are $\int_x \pi = \nu$ and $\int_y \pi = \mu$.

$$
\begin{aligned}
\mathbb{D}_W(\mu,\nu) &= \inf_{\gamma\in\pi}\iint c(x,y)\,\mathrm{d}\gamma(x,y)\\
&= \inf_{\gamma\in\pi}\iint c(x,y)\,\mathrm{d}\gamma(x,y) + \begin{cases}0, & \text{if } \gamma\in\pi\\ -\infty, & \text{otherwise}\end{cases}\\
&= \inf_{\gamma\in\pi}\{\iint c(x,y)\,\mathrm{d}\gamma(x,y) + \sup_{\varphi,\psi\to\mathbb{R}}[\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu - \iint(\varphi(y)-\psi(x))\,\mathrm{d}\gamma(x,y)]\}\\
&= \inf_{\gamma\in\pi}\sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \iint(c(x,y)-(\varphi(y)-\psi(x)))\,\mathrm{d}\gamma(x,y)\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \inf_{\gamma\in\pi}\iint(c(x,y)-(\varphi(y)-\psi(x)))\,\mathrm{d}\gamma(x,y)\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \begin{cases}0, & \text{if } \varphi(y)-\psi(x)\le c(x,y)\\ -\infty, & \text{otherwise}\end{cases}\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu\}.
\end{aligned}
$$

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *ICLR*, 2019.

Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *NeurIPS*. 2019.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. In *NIPS*, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Naveen Kodali, Jacob D. Abernethy, James Hays, and Zsolt Kira. How to train your DRAGAN. *arXiv:1705.07215*, 2017.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. *arXiv preprint arXiv:2209.15421*, 2022.

K. Kumar, Rithesh Kumar, T. D. Boissière, L. Gestin, Wei Zhen Teoh, J. Sotelo, A. D. Brébisson, Yoshua Bengio, and Aaron C. Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In *NeurIPS*, 2019.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J. Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *NeurIPS*, 2018.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, April 2022. arXiv:2204.06125 [cs].

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018. ISBN 0262039249.

Cédric Villani. *Optimal Transport: Old and New*. Springer, 2009 edition, September 2008. ISBN 3540710493.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.